# Machine Learning and Music Composition

Daniel Woeste
Division of Math and Science
University of Minnesota, Morris
Morris, Minnesota, USA 56267
woest015@morris.umn.edu

## ABSTRACT

Throughout history, musical composition has been thought of as limited to those gifted with a higher insight. As computers have risen in prominence, they have also asserted themselves as an almost necessary tool used in the field of music. Recent advancements in machine learning technologies has possibly provided a new way for computers to be used in the field of music, as well as possibly bringing music composition to the masses. Machine learning offers the unique possibility of having a program generate the song, leaving the user to edit or pick the song closest to their musical tastes.

## Keywords

Random Forests, Markov Chains, Machine Learning

## 1. INTRODUCTION

Music has become an almost daily part of our lives, whether it is through background in movies, commercials, or even listening to it for the sake of listening to a song. For many of us, our interactions with music are mostly a passive experience, listening rather than contributing. Recent advancements in machine learning may give people the chance to produce music tailored to their personal music tastes. These programs can be used as musical assistance to enhance or fill in gaps in musical knowledge. They can also be used to autonomously generate music without much input from the end user.

In this paper, I will cover three possible methods that may be used to generate music. These two methods are random forests, and markov chains explored by Ackerman et al. [1], and Klinger et al. [3] respectively. We will start the paper, in Section 2, by going over the necessary musical and machine learning terminology necessary to understand the larger concepts. From there we move into Section 3, where I describe exactly what the programs are and the general ideas behind them. After that we progress to Section 4, were we cover how the programs were built and trained to produce music. Finally, we will discuss the actual results of the research in Section 5.

## 2. BACKGROUND


*UMM CSci Senior Seminar Conference, November 2017* Morris, MN.

In this section, we discuss the necessary terminology both musically and that of machine learning to understand concepts throughout the rest of the paper.

### 2.1 Music

In this section we explain the terminology that is important to the musical side of this paper.

A melodic progression is the interval traveled when changing from one note to another. These intervals are broken down into whole steps and half steps. Combining several progressions together produces a melody. A melody is a combination of pitches, rhythms and note duration. At any point during a piece of music, the most important part played is considered the melody.

Chords are when several notes are being played at the same time. This can have a pleasing effect known as harmony, otherwise known as consonance, or a clashing effect known as dissonance. While most songs tend to be harmonic, dissonance gives the music drive and forward motion. Without dissonance, music can lack a sense of progression. Dissonance, in a simple sense, gives the music a direct path to follow, moving from a clash to peace and harmony

If a more in depth understanding of music is desired, please refer to *The Enjoyment of Music* by Forney et al. for a step by step build up of musical terminology and ideas [2].

### 2.2 General Machine Learning Framework

All of the methods described in this paper share a similar framework that would be useful to understand. The commonality between all the methods is that every note is generated in relation to a sequence of previous notes. Sometimes the new note is only dependent on a single previous note, while in other circumstances it depends on a larger set of notes. This method of relying on previous notes allows the program to generate new notes that fit well with the current melodic progression. Similar to how a sentence is formed, each new word is dependent on the previous word to make a coherent thought. A coherent melody is created by using notes that blend well with the overall theme already being expressed.

### 2.3 Machine Learning

Machine learning, a subcategory of artificial intelligence, is an area of computer science that focuses on programs that are able to systematically change their behavior. These learning systems are able to make alterations to their behavior. Training uses existing musical melodies. These melodies are used as a "final answer" that the programs are built towards. Using multiple passes over the training melodies, the

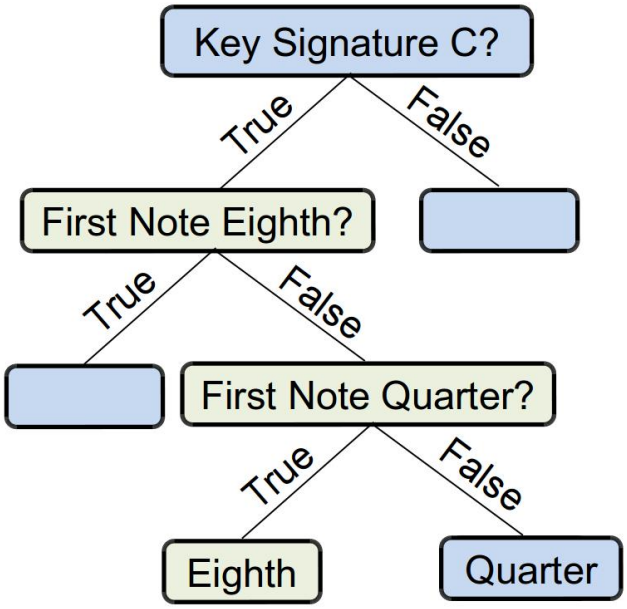


**Figure 1: Decision tree example**

program slowly approaches a correct answer by comparing new changes to the target melodies. After a set training time, the program is able to generate melodies that are similar in style, but not identical to the training data set. We will go over training in more detail in Section 4. This style of data driven learning allows machine learning programs to be well suited for applications such as self-driving cars, text prediction, online recommendations, and even the field of music composition.

In the remainder of this section we will describe other basics of machine learning including *decision trees*, *overfitting*, and *finite automata*.

### 2.3.1 *Decision Trees and Random Forests*

Decision trees are branching structures that represent tests and their possible outcomes, see Figure 1. In this example node, and the corresponding question being asked, are represented by rectangular boxes. The outcomes of these tests/questions are represented by the branches away from the nodes. In this case the outcomes are binary value, another possibility used in Figure 10 is to tie weights to the branches. The decision trees in this paper are read in a top down style, meaning that the tree in Figure 1 would be read by first asking if the piece of music was in the key of C. If answer to that question was a true statement, then we would progress through the tree down the branch indicating true. Following the tree the rest of the way through, we would reach the output of the tree, represented by the eighth and quarter values in the nodes at the lowest level of the tree [4].

Random Forests are a direct extension of decision trees, they are built using a multitude of decision trees. This results in there being many answers that can be selected for the final output. The problem of the multitude of answers is solved by taking a "vote" of the outputs of the decision trees. A distribution forms from this consensus, and the most common answer is picked as the final answer. For these trees to be built from a training data set, the training data needs

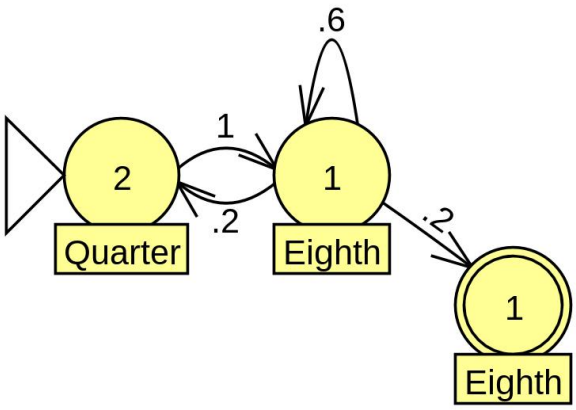


**Figure 2: Finite Automata made for an example rhythm in Klinger et al. [3]**

to be broken down into smaller chunks that can be used to train individual trees. The small sections of data are used to reduce overfitting of the decision trees by preventing one feature from becoming dominant.

### 2.3.2 *Overfitting*

A possible side effect with decision trees and all types of machine learning is overfitting. Machine learning algorithms and programs, such as decision trees, develop overfitting because the algorithm itself becomes too complex. Through multiple iterations of training, the algorithm develops too many parameters that are overly specific to the training data set. When this happens the program begins to learn and integrate small fluctuations in the data that are inconsequential to the actual outcome, instead of learning the overall "big picture." A learning system that has developed overfitting cannot be accurately run on data that is not the training data. Overfitting can be prevented by taking a system that has already been trained, and manually altering the program to remove the level of detail that it has. Musically this means that a program developed for composition would only be able to produce the exactly, or almost identical, pieces of music to what it was trained on. It would have failed to grasp the style of music, but instead would only be able to repeat exactly the music it has learned.

### 2.3.3 *Finite Automata*

Finite automata, otherwise known as finite state machines, are representations of an abstract computational model; see Figure 2. Finite state machines represent a stochastic model of the data. These state machines can only exist in one position at a time. Every possible move is represented by a probability, but the path through the automata is chosen at random, following the probabilities given. In Figure 2, we can see an automata that starts on a quarter note, this node has only one possible move, and that is a 100% chance to move to node labeled eighth. This node now has three possible directions that it could progress, it has 60% chance of returning to itself, and a 20% chance of either progressing back to initial node, or to the final node.

## 3. METHODS

Now that we have covered the necessary background information, we can now progress onto the two programs introduced earlier in the introduction. These two programs are *ALYSIA*, and *markov chains*.

## 3.1 ALYSIA

Automated LYrical Song-writing Application, ALYSIA, is a program that uses random forests to generate melodies for the lyrical inputs constructed by Ackerman et al [1]. These are built around an original set of songs that Ackerman et al. wanted to emulate, in this case, pop songs. Ackerman et al. used the data format Musix-XML, MXL, because it allows the random forests to pull ideas out of the training data. This process is known as training and we will go over it in section 4.1. Ackerman et al. chose random forests for their research because it allowed, after the fact, easy evaluation of why one note would be chosen after another. They are structured in a way that allows the user to read and determine the most important features from the trees. It is nearly impossible to do this for other methods, such as markov chains.

ALYSIA was developed to produce music for a given set of lyrics; these lyrics could be anything from a poem to an existing song, which we wish to recompose into a different style. It was also built as a system that functions in a co-creative nature, providing the ability to output multiple melodic options at once, allowing the user to insert their own musical tastes into the final output.

Ackerman et al. built ALYSIA around the idea of being able to extract features from the music and to use those features to help generate melodies. This feature extraction was made to look for patterns and other ideas of melodic, lyrical, or rhythmic importance to the target music. There were fifty-nine extractable features built into the program, including [1]:

- First Measure - A boolean variable indicating whether or not the current note belongs to the first measure of the piece
- Key Signature - The key signature that applies to the current note
- Time Signature - The time signature that applies to the current note
- Offset within Measure - The number of beats since the start of the measure
- Syllable Number - The syllable position within its word
- Scale Degree, Accidental, and Duration of 5 previous notes

These features are used to to determine the shape, potion, and lyrical importance of every note in the pieces of music used for training.

ALYSIA has several user input parameters that can change how the program produces its melodies. These parameters are known as *Explore/Exploit*, *Melody Count*, and *Rhythm Restriction* [1]. The first of these parameters, Explore/Exploit, is an integer parameter that determines how strictly the program follows the distribution of outputs from the random forest. Giving a higher value to Explore/Exploit the more the program will "exploit" the decision trees. This means that the program will be more likely to choose the most common output and follow the rules more. Conversely, with a lower value, the program will be more likely to explore musical ideas, more likely to stretch the rules and not choose the most common output [1].

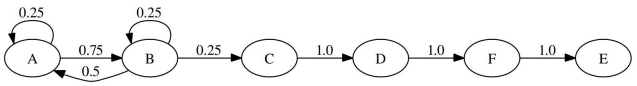


**Figure 3: Absolute Model [3]**

Melody Count is also an integer parameter. In this instance, the parameter determines the number of different melodies being produced at one time. This ties directly into the co-creative nature of ALYSIA, giving the user direct control of how many choices they will have at the end of process [1].

The last parameter, Rhythmic Restriction, is used to determine specific rhythmic values that the user would like to be remove from the selection process. For instance, if the the user would like to eliminate some of the less common notes, such as a dotted half note, they would then enter that value into Rhythmic Restriction. The value dotted half note would them be removed from the process entirely when generating a new set of melodies [1].

## 3.2 Markov Chains

Klinger et al. [3] made a program that uses markov chains to produce music. Markov chains differ from the earlier style, discussed in Section 3.1, in the way that the next note is determined. Markov chains use an entirely a probabilistic model to determine the next note value. Instead of asking a series of questions, the markov chain model used relies solely on the previous note and a set of probabilities to transition to the next note, as seen in Figure 2.

Similar to ALYSIA, Klinger et al. also broke down the generation of music into two categories, rhythm and pitch. The markov process follows a set pattern of events. First, a rhythmic pattern is developed. From there the markov chained used for rhythm is then run to produce a series of pitches. Once both of the models have been run, and we have series of rhythms and pitches, the two are then combined together to produce the final melodic output.

Due to the nature of pitch values having a name, and an interval the markov chain method broke down the ability to produce pitches into two different methods. These two methods are *absolute* and *relative* markov chains.The absolute model, seen in Figure 3, for the markov chain uses direct pitch values from the octave (CDEFGABC). This style of model allows for an easy to read output, because the markov chain is telling you directly what the note transitions are. The drawback to this is that it requires more configuration from the user beforehand. The user is required to specify the key and scale of that they would like the final piece to be in [3]. A melody generated by Figure 3 would start on an A, from there it would transition between staying on the A to moving to B and back. Once the transition to the C has occurred, it will complete the ascending line up to the E. If the markov chain needs to produce more pitch sequences after the E it will loop back to the first node, A, and start over.

On the other hand the relative model, seen in Figure 4,

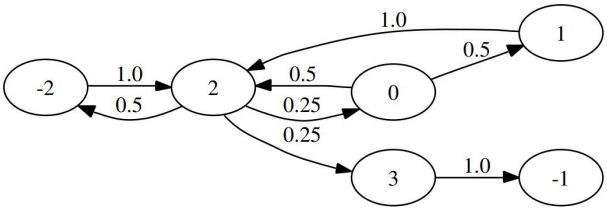


**Figure 4: Relative Model [3]**

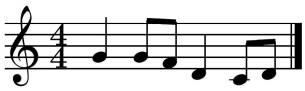

**Figure 5: A single bar melody produced after the second iteration of markov chains**

determines pitch changes by using interval lengths. By using the distance between notes rather than an absolute value of the note, this give the user much more freedom with the output of the relative pitch markov chain. By this we mean that the key, instrument, and position within the octave can be changed at anytime.

The last step to the markov model is to do post processing, known as mutation and recombination, uses the methods *one-point mutation*, *note splitting*, and *recombination*. The goal for each of these is to change the melodies made by the markov chain program, after the fact, in order to make it more complex and interesting [3].

The first of these post processing methods is known as one-point mutation. It is a function that takes the melodic output of the markov chains and traverses over it. As one-point mutation is traversing over the melody, every note that it encounters has a low probability of being raised or lowered in pitch value [3]. In Figure 6, we can see that the second note, from the left, has been raised up a half step. This is indicated by the sharp or # symbol. The method also changed the values of the fourth and fifth note in the sequence, it lowered them one step and two steps respectively.

Note-splitting is a process that is almost identical to the the previous operation, one-point mutation. The difference in the two operations is that instead of changing pitch, note-splitting changes the duration of the note [3]. In Figure 7, we can see that the second and third notes, both eighth notes, have been split into a run of sixteenth notes. The note durations have been cut in half.

The final style of post-processing is known as recombination. Recombination is a method of combining two separate melodies, into a new singular idea. It is working towards trying to introduce different melodic ideas into the same piece. For the markov chain program, this would mean that two

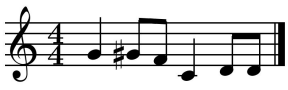

**Figure 6: Previous melody after one-point mutation operation**

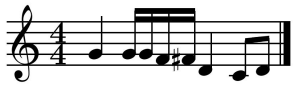

**Figure 7: Previous melody after note splitting operation**

different melodies would have to be generated. Recombination then takes these melodies and splices them together, without losing any of the original parent melody. The result is a new piece that is twice as long and that modulates between the melodic ideas.

# 4. TRAINING AND EVALUATION

Now that we have covered the concepts and ideas behind two methods for producing music, we can now move onto how these programs were trained to understand whether or not the music they produced is good or not. Training of these methods relies heavily on being able to evaluate the melodies after each generation.

## 4.1 ALYSIA

ALYSIA started with a training set of twenty-four pop songs, all consisting of a similar style. The specific songs were not listed by Ackerman et al. [1]. After having selected the songs to be used, they then needed to be converted to the correct file format used by ALYSIA, the file format is XML as mentioned earlier in Section 3.1. From there the feature extraction can then be applied to the freshly formated training data. Ackerman et al. states that of the twenty-four songs and fifty-nine extractable features they were able to make 12,358 observations on the songs that were used to build the final decision trees.

For the actual construction of the trees themselves, Ackerman et al. only states that they use R to build the trees for the random forests. No more information is given on the construction of the trees [1].

Once everything is complete and generated songs are ready to be evaluated, Ackerman et al. used the same set of feature extractions to evaluate the new melodies. They used feature extraction to compare the actual note, the note from the original melody used for training, to the produced note from the decision trees. This was done with the goal of creating random forests that would mimic the target style of the training data [1].

## 4.2 Markov Chains

The markov chain method approached the problem of choosing a training set a little differently than the ALYSIA method. Klinger et al. [3] started with the idea of introducing large amount of variance into the initial songs that were used to train the rest of the system. They specifically state that the initialization melodies should contain enough variance to produce innovative melodies. Otherwise, the program runs the risk of producing identical music to the training data [3].

Taking that idea in mind, Klinger et al. generated their own initial melodies by using random walks of notes and rhythms. They hoped that the random nature of the initial melodies would introduce the level of variance need to produce good innovative melodies [3].

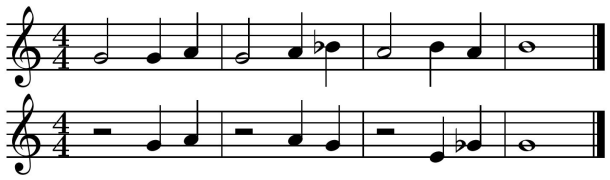

**Figure 8: *Rests on Downbeats* with a value of 0 for the first melody and 1 for the second [3]**

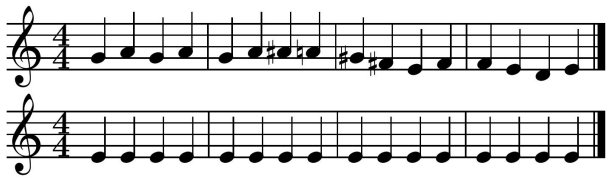

**Figure 9: *Repeated Pitch* with a value of 0 in the first melody and 1 in the second [3]**

Klinger et al. [3] tried several methods of evaluation. Two of these that we will talk about in this paper are *Feature Extraction* and *Decision Trees.*

Similar to the feature extraction described for ALYSIA, Klinger et al. is looking for ideas or patterns that could be important for determining the quality of the music. Instead of looking for specific musical parameters, such as key signature, the markov chain method made several functions that calculated ratios out of prominent features within the music. Two of these features are *rests on downbeats,* and *repeated pitch.*

Rests on downbeats determines the ratio between the number of downbeats and the number of downbeats on which there is a rest. A downbeat is the very beginning of the bar, notice in Figure 8 that the melody is broken into four sections. Each of these sections is separated by a vertical line. The beginning of each of the groups, from the left hand side, is the downbeat, it indicates the begging of the bar. A rest, indicated by the small black box on the middle line of the staff in the second example of the figure, indicates a break in the music, a place with no sound. In Figure 8 are two melodies with 4 downbeats: One with a value of 0 with no rests and one with 3 rests and a resulting value of 0.75 [3].

Repeated pitch calculates the ratio number of note transitions with interval zero and the number of notes. What this means is that, while moving through the music, every note that is the same as the next note has an interval of zero. Every note that is not that same has a non-zero interval. So, in Figure 9 in the first example, the first two notes are not the same, they have an interval of 1. While, in the second example the first two notes are the same so they have and interval of zero. In the example in Figure 9 is one melody without pitch repetition, value 0, and one with all possible pitch repetitions, value 1, [3].

Klinger et al. then tried an evaluation method using decision trees. Using the Weka-library in JAVA, Klinger et al. built a variety of decision trees with different numbers of fitness classes, otherwise known as final scores. The decision tree in Figure 10 has only two fitness classes, 0 and 10.

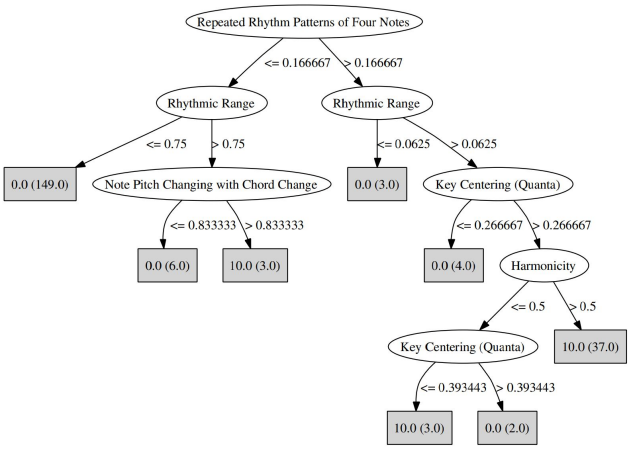


**Figure 10: Example of a decision tree used to evaluate melodic outputs for markov chains [3]. The ovals in this decision tree indicate the feature the is being evaluated, while the boxes indicate the final score.**

This tree was built as an extension to feature extraction. By providing the decision trees with these features, the decision tree is then able to string the feature values together to find series of questions that are important.

If the feature Repeated Rhythm Patterns of Four Notes is very small, <=0.16667, and Rhythmic Range is also not very high, <=0.75, the individual is classified as a bad melody, the left most box. But Repeated Rhythm Patterns of Four Notes is very small, <=0.16667, and Rhythmic Range is high, >0.75, and the Note Pitch Changing with Chord Change is also high, >0.8333, it is classified as being a good one, the third box from the left. Likely the following explanation holds: In the examples the chords are often changing with the bars. So if the rhythmic range is high there is a good possibility that the rhythm is confusing, but if there is always a note on the first beat in a bar it is considered not bad [3].

# 5. RESULTS

With there evaluation methods ALYSIA was able to achieve an accuracy of 86.79% for their rhythmic model. Ackerman et al. [1] derived this from a chart of all predicted notes made by ALYSIA. The table compared all of the predicted rhythmic values with what the expected value was. This gives a breakdown, for each note length, of how many were predicted correctly, and how many where predicted incorrectly. When a note was predicted incorrectly, the chart also indicated what the predicted note value was. Using this, Ackerman et al. calculated the average accuracy for all of the notes predicted to get their results. This chart showed that there were three major groupings of notes that were predicted by ALYSIA, sixteenth, eighth, and quarter notes [1].

When training towards their original set of twenty-four pop songs they were able to achieve an error of only 4.6% when the next note should have been an eighth note. Conversely, they had an error of 78.5% for note predictions when the next note should have been a dotted half note, the dot on the note adds length to it [1]. This points to the fact that the

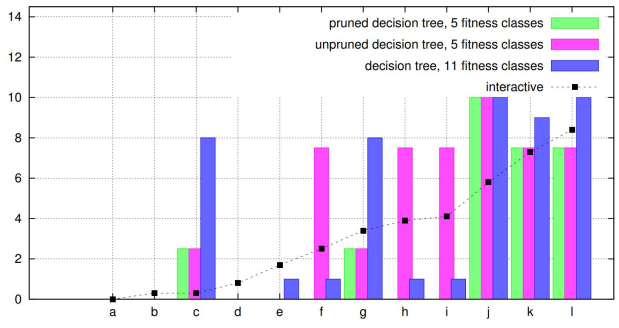


**Figure 11: Markov chain results plotting decision tree vs interactive evaluation [3]**

random forests were much better at predicting the average case rather than obscure fringe cases. Which is unsurprising, due to the fact that random forests should be better able to predict the cases that they encounter the most. There were also large groupings of predictions around non-dotted sixteenth, eighth, and quarter notes.

In the case of the pitch model ALYSIA was able to obtain an accuracy of 72.28% accuracy. Ackerman et al. used the same method breaking down and charting the predictions compared to the expected note to calculate the pitch accuracy as well. Unlike the rhythm model, the pitch model had large evenly spaced groupings of predictions throughout the octave. The error rate, depending on the note ranged between 18.5% to 42.1% for non-modified notes. Whereas the notes that had modifications, either a sharp or a flat to change pitch, generally had a much higher error rates, reaching 72.7% in one instance [1]. This indicates that the program was much better at predicting notes when they followed the original key signature of the piece, but experienced a drop off of accuracy when predicting notes that may not have fit as well with the flow of the melody. This loss of accuracy in the pitch model, compared to the rhythm model, may point towards an inherent increase in complexity when predicting pitch.

Klinger et al. [3] did not provide the same level of concrete data on their machine learning method as Ackerman et al. [1] did for ALYSIA. One method that Klinger et al. used was an interactive rating. A user would listen to the melodic output and would then be prompted to give a rating from 0 to 10 in increments of 0.1. They then compared these interactive results to the results of their other evaluation methods. We will look at the comparison between a few decision trees, as described in Section 2.3.1, and the interactive experience.

In Figure 11 we can see a comparison of three decision trees with different numbers of fitness classes and levels of pruning. Pruning is the process of removing section of the decision tree that are ineffective at classifying fitness. What this means is, nodes and branches are being removed from the tree that do not constructively add to reaching a score. This could be branches that have large amounts of repeated questions, or nodes that are inconsequential over all. The higher the number of fitness classes adds more granularity to the way the decision tree can grade the melodies. A decision tree with five fitness classes would still score from zero to ten, but with increments of two. The individuals $a$ to $l$ are sorted with respect to the interactive evaluation. The tree with eleven fitness classes makes some errors on the bad individuals and the unpruned one with five fitness classes is questionable in the middle fitness area. The pruned tree with five fitness classes leaves the best impression [3].

# 6. CONCLUSION

For ALYSIA, one of the main points for choosing random forests and decision trees was to extrapolate reasons behind the predictions of the forest. Pitch-wise, they found that being able to look back at the previous note was by far the most important feature. Where as, knowing the key signature of the piece proved about half as important as where the melody had just been. This indicates that knowing where the melody has just been can prove to be far more important than other factors such as key signature that would have been defined at the beginning of the piece. Interestingly, they found that being able to look back to the fifth previous note proved to be more important that looking back three or four notes [1].

Rhythmically speaking, Ackerman et al. [1] found that there was not a single feature that dominated everything like the pitch model. They did find that beat strength was the most important feature. Surprisingly, close behind in third position, key signature also played an important role in the production of rhythmic patterns. This was shocking because key signature, conventionally, deals solely with pitch values and has nothing to do with rhythmic.

For Klinger et al. [3], we can see that the tree that evaluated the most accurately, in comparison to the interactive response, was the pruned tree with five fitness classes. This may imply that a slight bit of human interaction during the training or at the end may improve performance. Also, there are a couple interesting fringe cases in Figure 11, mostly related to the eleven fitness class decision tree. For instance, melody C, in the previously mention figure, was scored very highly by the eleven fitness class tree, but very poorly by the interactive score. The same thing happens but in reverse for melodies H and I. This may point towards the fact that allowing more fitness classes results in a more complex tree that looses its ability to score correctly.

In conclusion, Ackerman et al. [1] were able to extrapolate some sense of reasoning behind the decision trees. Most of the important features followed a logical approach to producing music, but some of the important features ended up being more surprising. Klinger et al. [3] was able to produce melodies that rated very highly by both the decision trees and the interactive experience. Generally the interactive experience and the autonomous scoring agreed fairly closely on score, except for a few fringe cases related to the larger trees.

# 7. REFERENCES

[1] L. Ackerman. Algorithmic songwriting with alysia. *EvoMusArt*, pages 1–16, March 2017.

[2] M. Forney, Dell'antonio. *The Enjoyment of Music*. WWNORTON, New York City, New York, 12th edition, 2014.

[3] R. Klinger. Automatic composition of music with methods of computational intelligence. *WSEAS TRANS*, pages 1–16, March.

[4] Wikipedia. Decision trees - wikipedia, the free encyclopedia, 2017. [Online; accessed September 12, 2017].